UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

GeDisPro (Gestión y Distribución de Productos)

Gabriel Otra Fuster

Tutor: Francisco José Martínez Zaldívar

Trabajo Fin de Máster presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Máster en Ingeniería Telecomunicación

Curso 2019-20

Valencia, 24 de julio de 2020

Resumen

En estos últimos años y debido a una gran evolución tecnológica, se ha percibido un gran aumento en la creación de aplicaciones móviles. El desarrollo de estas aplicaciones cada vez se ve más dificultado debido a la gran cantidad de sistemas operativos y dispositivos que existen, cada cual ligado a un desarrollo en concreto. Debido a esto, siempre ha sido importante seleccionar la plataforma (sistema operativo y dispositivo) para la que desarrollar la aplicación. Sin embargo, el acto de tener que seleccionar solamente un sistema operativo para crear la aplicación es un inconveniente, ya que solamente se centra en una gama concreta y no puede abarcar una gran cantidad de mercado, teniendo así que ser desarrollada varias veces para poder cubrir la totalidad del mismo.

En la actualidad, existen varios *frameworks* que permiten desarrollar una aplicación y distribuirla en diferentes plataformas. Estos *frameworks* tienen la gran ventaja de poder ejecutar la aplicación independientemente del sistema operativo y del dispositivo utilizando un único código.

Este proyecto se centra en la creación de una aplicación que permite la gestión y distribución de productos para una empresa que proporciona productos químicos y de limpieza a hoteles.

El enfoque para el desarrollo de la aplicación contempla que pueda ser utilizada tanto en el lado del cliente como por la empresa propietaria. La aplicación constará de tres niveles de autenticación para cada tipo de usuario que permitirán: gestionar productos de almacén y pedidos de los clientes (administración), tener un seguimiento y una ayuda para realizar los repartos de los pedidos a los clientes (repartidores), facilitar la realización de los pedidos semanales o mensuales (clientes).

Resum Valencià

Aquests últims anys i degut a una gran evolució tecnològica, s'ha recaptat un gran augment en la creació d'aplicacions mòbils. La programació d'aquestes aplicacions cada vegada se veu mes dificultat degut a la gran quantitat de sistemes operatius i dispositius que existeixen, cada qual lligat a un llenguatge de programació en concret. Degut a açò, sempre ha segut important seleccionar la plataforma (sistema operatiu i dispositiu) per a programar l'aplicació. No obstant, l'acte de tindre que seleccionar solament un sistema operatiu per a la creació de l'aplicació es un inconvenient, ja que solament se centra en una gama concreta i no pot abastir una gran quantitat de mercat, tenint aixina que programar reiteradament la mateixa aplicació per a poder cobrir la totalitat del mateix.

En l'actualitat, existeixen diversos *frameworks* que permeten programar una aplicació i distribuir-la en diferents plataformes. Aquests *frameworks* tenen el gran avantatge de poder executar l'aplicació independentment del sistema operatiu i del dispositiu utilitzant un únic codi.

Aquest projecte se centra en la creació d'una aplicació que permet la gestió i distribució de productes per a una empresa que proporciona productes químics i de neteja a hotels.

L'enfocament per al desenvolupament de l'aplicació contempla que pugi ser utilitzada tant en el costat del client com per l'empresa propietària. L'aplicació constarà de tres nivells d'autenticació per a cada tip d'usuari que permetran: gestionar productes emmagatzemats i comandes dels clients (administració), tindre un seguiment i una ajuda per a realitzar els repartiments de les comandes als clients (repartidors), facilitar la realització de les comandes setmanals o mensuals (clients).

Abstract

In recent years and due to a great technological evolution, a great increase in the creation of mobile applications has been perceived. The development of these applications is increasingly difficult due to the large number of operating systems and devices that exist, each one linked to a specific development. Because of this, it has always been important to select the platform (operating system and device) for which to develop the application. However, the act of having to select only one operating system to create the application is a disadvantage, since it only focuses on a specific range and cannot cover a large amount of the market, thus having to be developed several times in order to cover the entire market.

Nowadays, there are several frameworks that allow the development of an application and its distribution in different platforms. These frameworks have the great advantage of being able to run the application independently of the operating system and the device using a single code.

This project focuses on the creation of an application that allows the management and distribution of products for a company that provides chemical and cleaning products to hotels.

The approach for the development of the application contemplates that it can be used both on the client side and by the owner company. The application will have three levels of authentication for each type of user that will allow: to manage warehouse products and customer orders (administration), to have a follow up and help to make the deliveries of the orders to the customers (distributors), to facilitate the weekly or monthly orders (customers).

Palabras clave

APP híbrida, Android, node JS, TypeScript, Plugins, App móvil, IOS, Electron, Html, multiplataforma, Sqlite, CSS, Apache Cordova, JavaScript

Índice

Tabla de ilustraciones

1. Introducción

El propósito de este documento es proporcionar la metodología y los pasos para crear una aplicación para una empresa encargada de abastecer de productos de limpieza y maquinaría industrial a los hoteles de la costa mediterránea.

1.1 Problema y Motivación

Actualmente existen muchas empresas, que se encargan de repartir mercancías o productos a comercios locales. Todas estas empresas, no suelen estar muy informatizadas debido a su tamaño, por tanto, siempre existen problemas en la pérdida de papeles, errores en los envíos o envíos que no se realizan en las fechas acordadas debido a errores humanos.

Tras trabajar para la empresa, la cual va a hacer uso de la aplicación que he confeccionado, me pude dar cuenta que algunos de los errores mencionados anteriormente y otros muchos como problemas internos en la gestión de productos en el almacén propio de la empresa, podían ser subsanados informatizando la gestión interna de la empresa.

1.2 Objetivo

El objetivo principal consiste en la creación de una aplicación multiplataforma, la cual servirá para hacer mucho mas fácil y amena la gestión de los pedidos de la empresa y en un futuro proporcionar información para una gestión inteligente del almacén.

Para la creación de dicha aplicación se utilizará el *framework* Ionic, uno de los más utilizados a día de hoy para la creación de las nuevas aplicaciones del mercado. Además de este *framework* se utilizará Node.js y Sqlite como base de datos.

La aplicación que se va a desarrollar contendrá una base de datos en la cual se encontrará una gran lista de los productos proporcionados por la empresa.

Una parte de esta lista de productos será vista por cada cliente, para que puedan hacer sus pedidos, y del mismo modo desde la empresa puedan modificar y administrar, tanto los pedidos como los productos de los que disponen.

Como objetivos personales, este proyecto me ayudará a aumentar mis conocimientos en el ámbito de las aplicaciones informáticas y aportará beneficios a mi carrera profesional aprendiendo nuevas tecnologías.

2. Estado del arte

2.1 Introducción

Hace algunos años el desarrollo de aplicaciones nativas era la forma más eficiente de obtener una aplicación funcional para el dispositivo deseado. Esto ha ido cambiando en el transcurso del tiempo, en los últimos años el desarrollo de las aplicaciones híbridas multiplataforma ha aumentado hasta el nivel de ponerse en el mismo nivel del desarrollo de las aplicaciones nativas. Todo esto ha sido posible gracias a las mejoras en las funcionalidades y la aparición de nuevos *frameworks* mejorados basados en SDKs nativas.

Hoy en día, Ionic es uno de los *frameworks* de desarrollo de aplicaciones híbridas más importantes del mercado. Unido a Node.js para la gestión de *plugins* y la utilización de Apache Cordova para proveer las SDKs y librerías necesarias para una interacción nativa.

En la actual era de la tecnología digital, las aplicaciones híbridas, son aplicaciones diseñadas en un lenguaje de programación web ya sea HTML5, JavaScript o CSS, junto con un *framework* que adapta la vista web a cualquier vista de cualquier dispositivo móvil. Por tanto, el implementar una aplicación con estas tecnologías, nos proporciona que la aplicación sea tanto adaptada para terminales iOS, Android o Windows Phone, evitando la creación de una aplicación para cada dispositivo.

2.2 Ionic Framework

Ionic Framework es un *software* gratuito, *open source* que, junto con Angular, permite el desarrollo de aplicaciones híbridas creadas en HTML5, CSS y JS.

Figura. 1 Icono Ionic

Ionic está basado en Angular, aunque ofrece una versión diferente dependiendo de la versión Ionic utilizada, ofrece la oportunidad de crear código eficiente, rápido y escalable. El *framework* tiene un uso sencillo y fácil de entender para cualquier desarrollador de aplicaciones, ya que esta basado en las SDKs nativas de las diferentes plataformas móviles.

Otra característica de Ionic es que cuenta con su propio CLI (*Command-Line interface*) por lo que, con el lanzamiento de un comando, se pueden crear, modificar, testear o compilar las aplicaciones para cualquier plataforma.

2.3 Angular

Figura. 2 Icono Angular

Angular es un *framework* Javascript respaldado hoy en día por Google. El modelo que usa Angular es MVC (Modelo Vista Controlador). Angular permite comunicarnos con el *backend* y modificar nuestro *frontend*. La principal ventaja de Angular es el gran modularidad que consigue con el uso de componentes. Los bloques de construcción básicos son NgModules, que recogen el código relacionado en conjuntos funcionales; una aplicación Angular se define por un conjunto de NgModules.

- NgModule: son el módulo raíz para iniciar un proyecto. Dentro de una aplicación se pueden importar varios módulos, esto permite tener el código organizado y tener una reusabilidad de los módulos en diferentes pantallas, esto permite facilitar el mantenimiento de aplicaciones complejas.
- Components: cada componente está asociado un trozo de la vista de la aplicación, el componente define variables y métodos que están disponibles en su *template*. Cada proyecto tiene al menos un componente raíz que conecta todos los componentes con el DOM (Modelo en Objetos para la Representación de Documentos)
- Template: el *template* es lo que se usa para definir la vista de un *component*, explicándolo de otras maneras, es un HTML pero con unas expresiones anidadas de Angular, que permiten aumentar las funcionalidades y modificar el comportamiento del *template*.
- Services: los servicios se usan para compartir información entre los componentes.
- Data Binding: esto lo que nos permite es enlazar datos en la aplicación bidireccionalmente, coordinando datos que existen en el *front* con su correspondiente *backend*.

2.4 TypeScript

Figura. 3 Icono TypeScript

TypeScript es uno de los lenguajes de programación de alto nivel que implementa mecanismos habituales de programación orientada a objetos, aportando una gran escalabilidad en las aplicaciones muy grandes.

Su característica principal es que compila en JavaScript nativo, por lo que se puede usar en los mismos proyectos que se usa JavaScript, por tanto, TypeScript es un "superset" de JavaScript aportando herramientas beneficiosas a los proyectos.

El tipado estático en Typecript es opcional, pero su uso es recomendado ya que aporta facilidad en el trabajo y en la depuración de los programas.

2.5 Sqlite

Figura. 4 Icono SQLite

SQLite es una biblioteca de software que proporciona un sistema de gestión de bases de datos relacionales. El "lite" en SQLite significa peso ligero en términos de configuración, administración de la base de datos y recursos necesarios.

SQLite es un software autónomo, que no necesita un servidor, la configuración no es necesaria y transaccional.

SQLite utiliza tipos dinámicos para las tablas. Esto significa que puede almacenar cualquier valor en cualquier columna, independientemente del tipo de datos.

SQLite permite que una sola conexión a la base de datos acceda a varios archivos de la base de datos simultáneamente. Esto aporta muchas características agradables como la unión de tablas en diferentes bases de datos o la copia de datos entre bases de datos en un solo comando.

SQLite es capaz de crear bases de datos en memoria que son muy rápidas de trabajar.

2.6 GitLab

Figura. 5 Icono GitLab

GitLab es un repositorio donde se puede almacenar el código fuente de forma pública o privada, y mediante el versionado de *git*, permite trabajar a nivel colaborativo en varios proyectos con distintos desarrolladores.

El punto más fuerte de GitLab es que se puede instalar en un servidor gratuitamente.

Es posible compartir proyectos con otros desarrolladores o invitarlos a contribuir a tu propio proyecto, esto es un beneficio a la hora de escalabilidad y equipo.

Al poder crear proyectos públicos y privados, las empresas con códigos privados o sensibles, pueden crear un proyecto privado y asignar a los desarrolladores que quieran que accedan al código.

3. Metodología, recursos y plan de trabajo

3.1 Metodología Ágil

Las metodologías ágiles son aquellas que permiten crear una forma de trabajo adaptándose a las condiciones del proyecto, permitiendo amoldar el desarrollo y conseguir una flexibilidad según las circunstancias del entorno.

Las empresas que trabajan con este tipo de metodologías consiguen aumentar la productividad y gestionar sus proyectos de forma mas autónoma, flexible y eficaz.

3.2 Definición del modelo

Dentro de las metodologías ágiles he decidido basarme en un modelo Kanban. Esta metodología consiste en la creación de pequeñas tareas y tenerlas en un cuadro creado con columnas de tareas; pendientes, en proceso, terminadas, etc. Las ventajas que proporciona esta metodología son:

- Planificación del proyecto en tareas más pequeñas.
- Si existen varios desarrolladores mejora el rendimiento de trabajo en equipo, ya que todos los desarrolladores pueden ver en que estado están las tareas y a quien le pertenecen.
- Normalmente la entrega de las tareas es continua, por tanto, el desarrollo de la aplicación va avanzando completando funcionalidades nuevas.

Hoy en día existen muchas Webs que proporcionan herramientas para poder crear las tablas Kanban gratuitamente, como Atlassian, que tiene una versión gratuita con las funcionalidades necesarias para hacer un proyecto con menos de 10 desarrolladores.

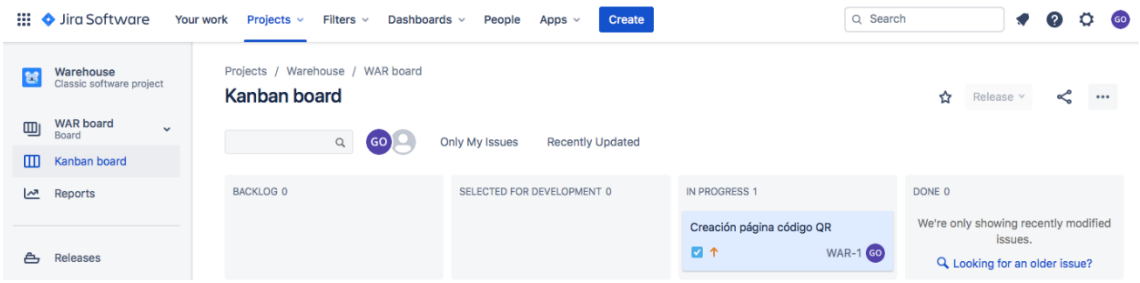

Figura. 6 Board Kanban

3.3 Aplicación del modelo

A la hora de aplicar la metodología al proyecto, lo mas importante es dividir todas las funcionalidades del proyecto en tareas más pequeñas, teniendo en cuenta que tras la entrega de cada tarea la aplicación debe continuar compilando y funcionando correctamente.

La aplicación de este modelo, definiendo plazos de entrega imaginarios (Sprints) y subdiviendo correctamente la aplicación en tareas lo más simples posibles, me han permitido compaginar correctamente mi vida profesional con el desarrollo del proyecto en cuestión.

Lo primero para aplicar el modelo, era entender correctamente todos los actores que pertenecen a la empresa y que tareas suelen realizar esos actores en su día a día. Por tanto, lo primero que hice fue dividir el proyecto subproyectos más pequeños dependiendo de los actores:

- Administrador
- Comercial
- Repartidor
- Cliente
- Mecánico

Una vez identificados todos los actores y las acciones que hacen diariamente, creé tareas para definir las acciones de los actores, por ejemplo, una persona de la parte de Administración se encarga de gestionar los productos que se tienen en el almacén, por tanto, una vez creada la base de datos y el servidor que fueron las dos tareas principales, en la parte de Administrador se crearon las tareas de "Crear productos", "Modificar productos" y "Eliminar productos", de esta manera tenemos 3 tareas simples fáciles de hacer, tras las cuales la aplicación continuaría operacional y con una nueva funcionalidad.

Normalmente los Sprints declarados no excedían las dos semanas, teniendo cada uno en su interior una cantidad diferente de tareas, dependiendo normalmente de la dificultad de las tareas a completar.

3.4 Recursos necesarios

En este apartado, se especificarán los recursos y los equipos que han sido utilizados para el desarrollo del TFM, así como para realizar todas las pruebas de funcionamiento.

3.4.1 Hardware

- Equipo de desarrollo (sobremesa)
  - CPU: 1,4 GHz Intel Core i5
  - Gráfica: Intel HD Graphics 5000 1536 MB
  - RAM: 4 GB 1600 MHz DDR3

- Dispositivo físico de pruebas (Smartphone)
  - Modelo: Samsung Galaxy J6
  - CPU: Exynos 7870 1.6GHz
  - RAM: 3GB/4GB
  - OS: Android 10

3.4.2 Software

- Sistema operativo
  - macOS High Sierra v10.13.6

- Entornos de desarrollo
  - Visual Studio Code
  - Android Studio
  - DB browser for SQLite

- Editores de texto
  - Microsoft Word

- Dispositivo emulado de pruebas (AVD Android Virtual Device)
  - Modelo: Android Emulator Pixel 2
  - CPU: Google Play Intel Atom (x86)
  - Pantalla: 1080 x 1920 pixels @ 420ppi.
  - OS: Android 8.1 (Oreo)

3.4.3 Plan de trabajo

Se ha escogido un plan de trabajo que se adecue perfectamente al modelo Kanban. Por tanto, se ha dividido el trabajo en diferentes fases las cuales se detallan a continuación:

1) Fase de documentación de tecnologías utilizadas:
   a) Visual Studio Code
   b) Ionic Framework
   c) Apache Cordova
   d) SQLite
   e) Typescript
2) Fase de análisis:
   a) Análisis de forma de trabajo y actores que intervienen
   b) Diseño de base de datos y servidor
   c) Diseño de las tareas para cada actor
   d) Diseño de la aplicación
   e) Implementación de la aplicación
3) Fase de pruebas
   a) Pruebas y tests en dispositivos
   b) Búsqueda y corrección de bugs
4) Reunión con el cliente
   a) Presentación de aplicación ante el cliente
   b) Fase de remodelado y correcciones (si es necesario)
   c) Fase de pruebas si se han hecho correcciones
5) Defensa del TFM
   a) Escritura de la memoria
   b) Creación de la presentación PowerPoint
   c) Preparación para la exposición

4. Análisis

4.1 Arquitectura y partes de la infraestructura

4.2 Casos de uso

El modelo de casos de uso como su nombre indica, nos especifica las acciones o las funciones las cuales puede realizar cada actor con la aplicación. Este tipo de modelos se ha vuelto muy importante en el diseño de software, ya que sabiendo que funcionalidades se quieren asignar a los usuarios se pueden crear interfaces o componentes los cuales se podrán reutilizar en el desarrollo de la aplicación siempre y cuando las funcionalidades sean parecidas entre si.

4.2.1 Actores

Los actores son los tipos de usuarios los cuales tendrán acceso a la aplicación y estarán ligados a unas funciones las cuales les permitirán hacer correctamente y más rápidamente su trabajo. En nuestro caso tendremos 5 actores diferentes, los cuales estarán identificados dentro de la base de datos en la tabla "role". Cuando un usuario se autentica, en la tabla "users" existe una 'foreign_key' que hace referencia a la tabla "role" y de ese modo obtenemos el rol de cada usuario que entra en la aplicación de ese modo, se le adapta el Menú de la aplicación con las funcionalidades necesarias.

4.2.2 Casos de Uso

Los casos de uso son los eventos que se definen entre los actores y la aplicación para alcanzar un propósito. En nuestro caso el actor utilizará la aplicación para realizar un pedido a una empresa, la cual proveerá de artículos a este cliente. Pero este proceso tan simple, requiere una serie de acciones a realizar por cada uno de los actores y en un orden concreto.

Como se ha explicado anteriormente, el usuario que se conecta a la aplicación puede tener varios roles y dependiendo de cada rol, podrá usar unas funciones o otras.

En la tabla inferior se podrán ver todos los usuarios y las funcionalidades asociadas a estos.

Administrador	• Creación, administración y supresión de usuarios. • Creación de pedidos. • Revisión y modificación de pedidos. • Asignación de pedidos a un Repartidor. • Modificación de productos del almacén. • Creación de códigos QR.
Comercial	• Creación y supresión de fichas de los clientes. • Revisión de pedidos.
Repartidor	• Revisión del pedido. • Cambiar orden de pedidos y consultación de ruta en el mapa. • Sistema de notificaciones
Cliente	• Creación de pedidos. • Notificaciones del estado del pedido.

Mecánico	• Creación máquinas y partes de máquinas. • Creación de incidencias sobre las máquinas. • Creación de códigos QR.

Tabla 1 Actores y funciones

También existen otras funcionalidades comunes a todos los usuarios, como el sistema de autentificación, donde el usuario podrá autenticarse de manera normal mediante el correo electrónico y su contraseña. También dispondrán de el servicio de "Olvidó la contraseña" el cual les enviará un correo con el cambio de contraseña correspondiente.

4.3 Modelo de Base de datos

Como se ha mencionado anteriormente, para este proyecto se ha utilizado la base de datos Sqlite, un tipo de base de datos relacional, la cual nos proporciona funcionalidades las cuales nos permiten acceder a todas las tablas y tener una relación, por ejemplo, entre los usuarios y su tipo de rol, o los pedidos que tengan hechos con *querys* simples.

Este tipo de bases de datos nos permite una mayor integridad de datos y una mayor seguridad mediante restricciones en operaciones o acciones.

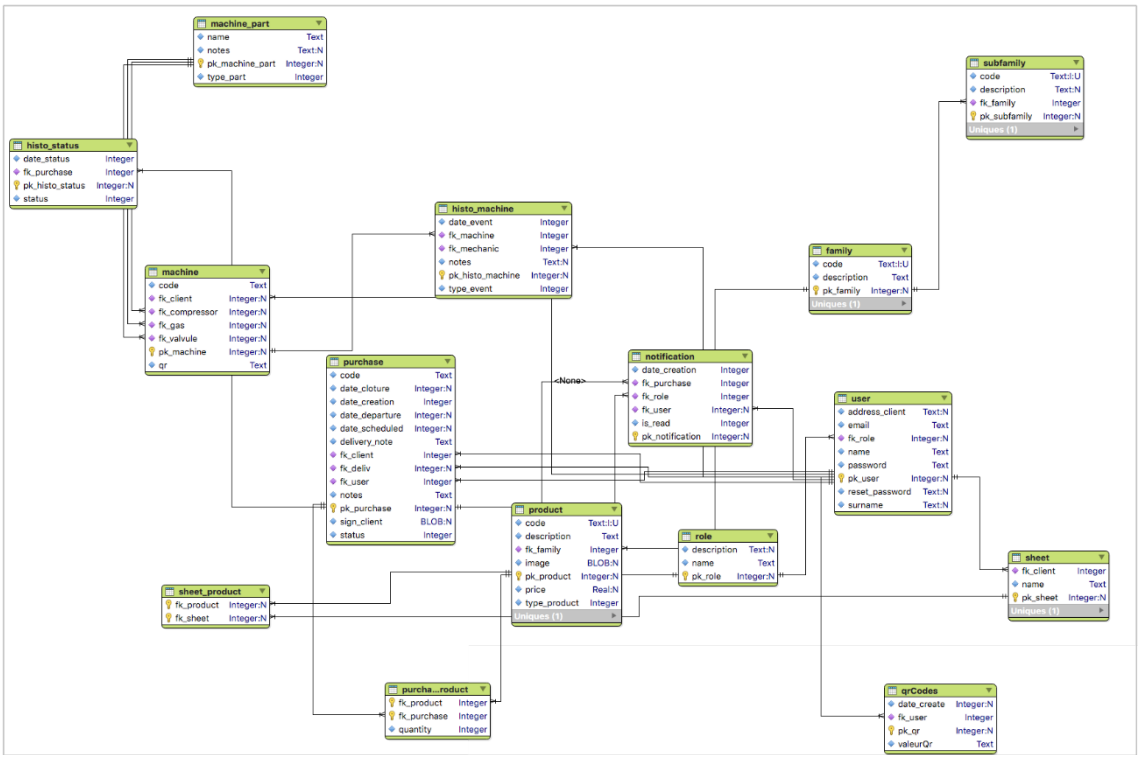

Figura. 7 Modelo BDD

5. Diseño

5.1 Pantalla de *login*

Es el punto de entrada a la aplicación. En esta pantalla se encontrarán dos campos de texto para insertar el email y la *password*, correspondiente a cada usuario.

Si el usuario no recuerda su contraseña, podrá acceder a la pantalla de olvidé *password*, mediante el botón "OLVIDE CONTRASEÑA".

Figura. 8 Pantalla Login

5.2 Recuperación de *Password*

En la pantalla de Reset Password se encontrará un campo de e-mail, donde el usuario deberá escribir el correo del que desea recuperar la contraseña. Dicho usuario recibirá un email con las instrucciones para resetear su contraseña.

Hay que tener en cuenta que, si el e-mail insertado no existe en la base de datos, se mostrará un mensaje de error en pantalla y, por tanto, no se enviará ningún mail.

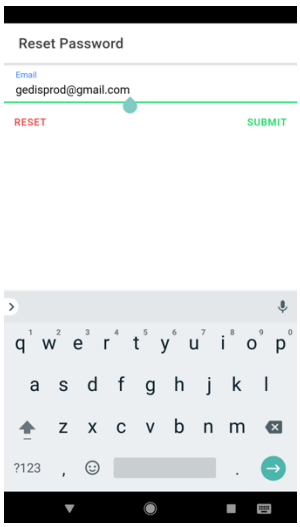

Figura. 9 Pantalla recuperación de password

5.3 Pantalla *management*

Es la pantalla principal de la aplicación. En ella se podrán encontrar varias entradas de un menú.

Las opciones de menú que se podrán visualizar dependerán únicamente del rol del usuario que la está utilizando. Por ejemplo, un administrador podrá ver el menú completo, mientras que otro usuario con el rol comercial, solo podrá observar la pestaña de creación de fichas.

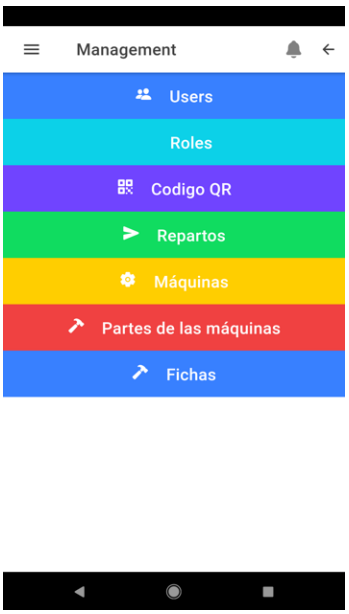

Figura. 10 Pantalla *management*

5.3.1 Menú lateral

Se ha programado un menú vertical, el cual se desplegará arrastrando el dedo desde la parte izquierda de la pantalla o pulsando el icono superior.

Dentro de este menú se podrán observar dos entradas: la de Management, que devolverá al usuario a la pantalla principal y cerrará el menú, y otra pestaña, la cual permite acceder a la creación de los pedidos.

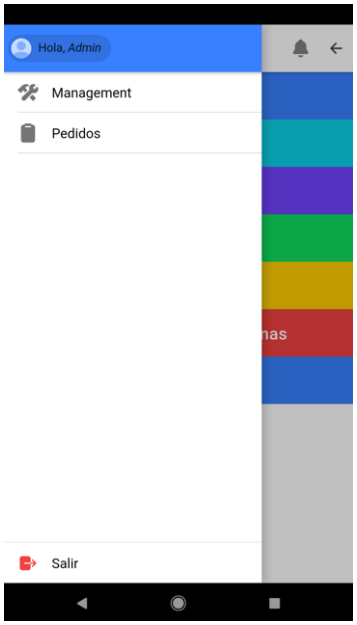

Figura. 11 Menú lateral

5.4 Pantalla creación de usuarios

La creación de usuarios solamente podrá hacerse desde la parte administrativa de la empresa, restringiendo así el acceso a usuarios comunes que no hayan tenido contacto con ninguno de los trabajadores de la empresa proveedora.

Las pantallas de creación de usuarios, máquinas, roles … tienen el mismo formato. En primera estancia, cuando se accede, se podrá observar una pantalla con una lista de todos los objetos (usuarios, máquinas…) que tenemos actualmente en la BDD.

En cada entrada de la lista existirá un botón para poder suprimir cada objeto creado.

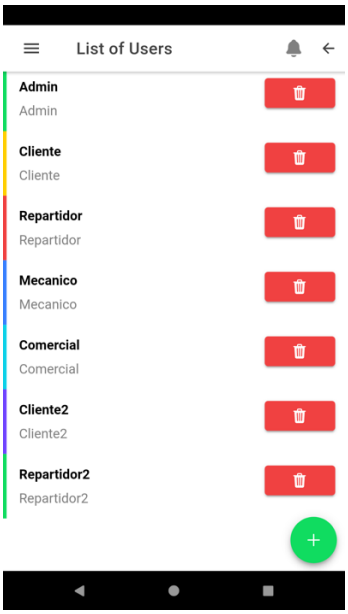

Figura. 12 Pantalla lista usuario

Mediante el botón verde situado en la parte inferior derecha de la pantalla se podrá acceder a la página de creación de los objetos. En ella se encontrarán distintos componentes (como InputsTexts o Dropdowns) para insertar los parámetros básicos para la creación de objetos.

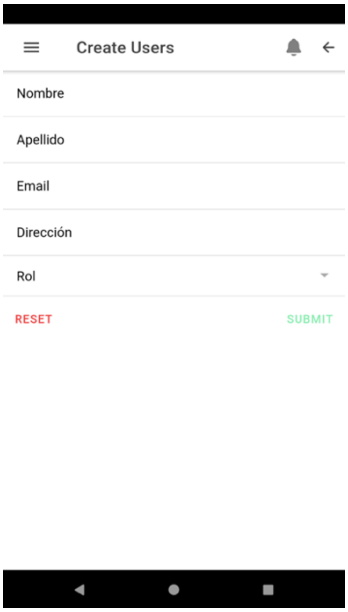

Figura. 13 Pantalla creación de usuarios

5.5 Pantalla creación de fichas

Las fichas son creadas por los comerciales o por los administradores. Estas son creadas siguiendo las necesidades de los clientes. Por ejemplo, un gran Hotel de la costa blanca, necesitará crear varias fichas, ya que los productos que necesitaría una cocina, no son los mismos que los productos que usa la lavandería.

Por tanto, un solo cliente puede tener varias fichas y deberá hacer varios pedidos. Esto, aunque parezca un inconveniente no lo es, ya que las personas encargadas en el lado del cliente pueden no ser las mismas para cada parte del Hotel.

Los productos de la empresa están clasificados por familias, por tanto, en la creación de las fichas se escogerá una familia, y una vez escogida la familia se podrán escoger los productos asociados a dicha familia.

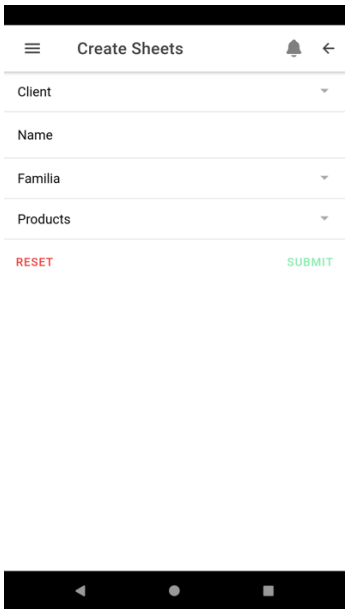

Figura. 14 Pantalla creación de fichas

5.6 Pantallas de pedidos

5.6.1 Listado de pedidos

En esta pantalla se encuentra un listado con los pedidos de cada cliente.

Si el usuario que está conectado a la aplicación es un cliente, en la lista de pedidos solamente aparecerán los pedidos de dicho cliente. Por otro lado, si es el administrador el que usa la aplicación aparecerán todos los pedidos de la base de datos. Finalmente, un repartidor solo podrá ver los pedidos que tiene asignados.

Los pedidos tendrán diferentes colores, dependiendo del estado del mismo, para facilitar su identificación:

- Creados: gris
- A repartir: amarillo
- En reparto: azul
- Entregado: verde
- Entregado con retraso: rojo

Además, el usuario puede filtrar los distintos pedidos por estado (o por cliente, en el caso del repartidor) o, incluso, buscar su código.

Figura. 15 Pantalla lista de pedido

5.6.2 Creación del pedido

La creación de un pedido solo podrá realizarse por el cliente o los administrativos de la empresa (administradores). A la hora de crear el pedido irán apareciendo *pop-ups*, en los cuales se proporcionarán informaciones necesarias para el pedido.

Se podrá adjuntar una observación (de forma opcional), escoger una ficha la cual filtrará los productos de la BDD para poder hacer el pedido solo con los productos declarados en la ficha.

Una vez escogida la ficha, se seleccionarán los productos que se quieran demandar y tras la confirmación se podrá añadir la cantidad que se desea pedir de cada producto. Finalmente, se mostrará un resumen del pedido antes de ser confirmado. Una vez aceptado, el cliente recibirá un correo con la confirmación y el resumen de su pedido.

Se podrán consultar todas las pantallas y el flujo que se sigue para crear un pedido en ANEXO I – Flujo de creación de pedido

5.6.3 Asignación de repartidor

Una vez creado el pedido, los administradores recibirán una notificación con un pedido nuevo. En tal caso, se procede a abrir el pedido realizado por el cliente y asignárselo a un repartidor. Tras asignarlo al repartidor se tendrá que establecer una fecha y hora de entrega del pedido. El estado del pedido cambiará de "Creado" a "A repartir". Una notificación con el nuevo pedido a repartir le será enviada al repartidor asignado.

En este nuevo estado (y en el anterior), los administradores también pueden, deslizando el ticket del pedido hacia la izquierda, modificar o eliminar el pedido.

Se podrán consultar las pantallas y flujo de asignación en ANEXOS I - Flujo de asignación de repartidor, fecha y hora

5.6.4 Pantalla reparto de un pedido

Cuando un pedido alcanza el estado "A repartir", el repartidor al que se le ha asignado el pedido recibe una notificación. El repartidor comprueba los pedidos asignados y organiza su reparto. Cuando el pedido esté listo para ser enviado al cliente, el repartidor preparará el pedido y el estado de éste pasará a "En reparto".

Figura. 16 Pantalla de reparto

5.6.5 *Pantalla de firma del pedido*

Una vez entregado el pedido, y revisado el albarán por el cliente para certificar que todo esta correcto, se necesitará la aprobación del cliente.

Esta pantalla permite al cliente firmar el albarán. La firma será añadida al albarán mediante la creación de un *pdf*.

Tras la firma y la confirmación del pedido, se mandará automáticamente un correo con el albarán firmado a la empresa y al cliente para dejar constancia que el pedido ya ha sido entregado.

El estado del pedido cambiará, por última vez, a "Entregado", si éste se ha entregado en la fecha y hora indicada o a "Entregado con retraso", si el pedido se entrega después de la fecha y la hora indicada.

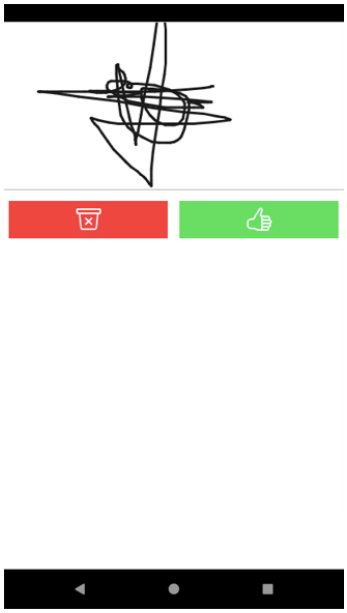

Figura. 17 Pantalla firma del cliente

5.6.6 *Estados del pedido*

Es una pantalla que suele tener las entradas que indican los estados por los que ha pasado el pedido y en que fechas y horas se ha cambiado el estado del pedido.

Figura. 18 Pantalla estados del pedido

5.7 Pantalla de repartos

La pantalla de reparto solo es accesible mediante el rol de repartidor. En esta pantalla se mostrarán los pedidos con el estado "En reparto" que el repartidor entregará ese día. El repartidor podrá ver en un mapa, mediante un *marker*, la dirección donde debe entregar cada pedido. Este *marker* está representado por un número el cual indica el orden de reparto de los diferentes pedidos. Además, en la parte inferior de la pantalla, el repartidor podrá reordenar sus pedidos para establecer el orden de reparto que considere oportuno.

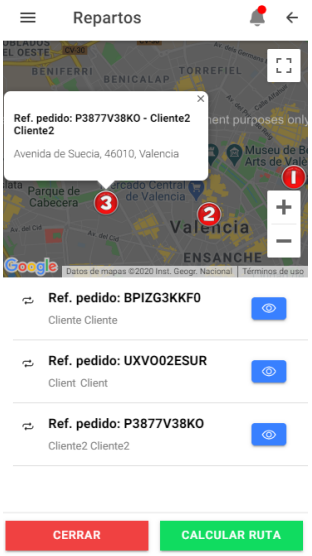

Figura. 19 Pantalla de repartos

5.8 Notificaciones

Para las notificaciones se ha instaurado dos tipos diferentes: las notificaciones dentro de la aplicación mientras se hace uso de ella y las notificaciones *push* para cuando la aplicación está corriendo en segundo plano.

Con respecto a las primeras, un usuario puede acceder a ellas mediante un icono de una campana, el cual mostrará un punto rojo cuando existan notificaciones nuevas. Dichas notificaciones aparecen cuando un pedido se ha creado, se ha asignado a un repartidor o se ha entregado a un cliente.

La pantalla de notificaciones se divide en dos partes: los mensajes no leídos, en la que el usuario también podrá copiar el código del pedido para buscarlo más tarde, y el histórico de mensajes, donde se almacenan todas las notificaciones.

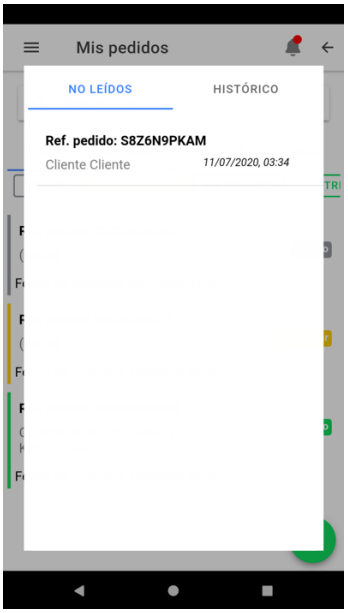

Figura. 20 Notificación interna

Mediante las notificaciones *push* el usuario podrá observar en su terminar que tiene un nuevo aviso de la aplicación.

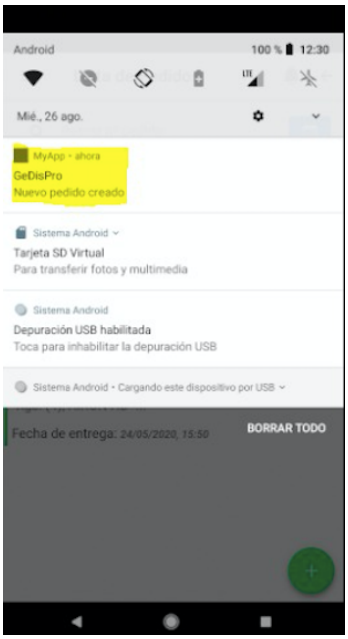

Figura. 21 Notificación push

6. Implementación

Como se ha mencionado anteriormente para la implementación se ha hecho servir el IDE VSCode al cual se le pueden añadir una serie de extensiones, las cuales nos permiten mejorar y acelerar la escritura de funciones y códigos. Para la implementación nos hemos hecho servir del *framework* Ionic, ya que nos permite la realización de aplicaciones híbridas mediante diferentes lenguajes y tecnologías. Los lenguajes utilizados han sido HTML5, CSS3 y TypeScript.

6.1 Server

En la parte del servidor se podrán apreciar dos partes, la primera será la base de datos SQLite la cual contendrá todos los datos que trata la aplicación.

La segunda parte, serán los códigos TypeScript los cuales contendrán las querys necesarias para acceder a las tablas de la BDD. Todos estos códigos serán los encargados hacer de puente con los servicios creados en la parte Front de la aplicación para poder extraer los datos de la BDD y mostrarlos por pantalla.

6.1.1 Implementación Controladores

Hoy en día las aplicaciones van evolucionando constantemente, por tanto, para la implementación de los controladores y las interfaces del servidor, se ha desarrollado un Script el cual, selecciona la base de datos y hace una *query* a *sqlite_master*, que es donde se encuentra toda la información de la base de datos, se buscan las tablas, vistas y se excluyen todas las tablas "sqlite_XX". Para tabla se extrae la información como el nombre de la tabla, sus columnas y tipos, y automáticamente crea los controladores con las *querys* necesarias en su interior para poder hacer las *querys* SQL básicas.

```
constructor() {
    const querys = {
        list: 'SELECT * FROM ${table}',
        create: 'INSERT INTO ${table} (${getRows(rows)}) VALUES (${getNQuestions(rows.length - 1)})',
        getOne: 'SELECT * FROM ${table} WHERE pk_${table} = (?)',
        update: 'UPDATE ${table} SET ${getRows(rows, true)} WHERE pk_${table} = (?)',
        delete: 'DELETE FROM ${table} WHERE pk_${table} = (?)'
    }
    super('${table.replace('_', '-')}', querys);
    this.querys = querys;
}
```

Figura. 22 Script GenerateControllers.ts

De esta manera si se añade una columna nueva en una tabla, no será necesario, hacer los cambios a mano, sino que lanzaremos el script y remplazaremos los controladores, de ese modo solo se tendrá que modificar la parte *front* de la aplicación.

Un ejemplo del resultado del lanzamiento de este script es el siguiente:

```
constructor() {
    const querys = {
        list: 'SELECT * FROM role',
        create: 'INSERT INTO role (name, description) VALUES (?, ?)',
        getOne: 'SELECT * FROM role WHERE pk_role = (?)',
        update: 'UPDATE role SET name = (?), description = (?) WHERE pk_role = (?)',
        delete: 'DELETE FROM role WHERE pk_role = (?)'
    }
    super('role', querys);
    this.querys = querys;
}
```

Figura. 23 RoleController.ts

6.1.2 *Mail Sender*

Se ha implementado la función *sendMail* que hace uso de la librería *nodemailer* para enviar correos electrónicos de una manera sencilla.

Para el desarrollo, se ha usado *gmail* como servidor SMTP, para hacerlo sencillo se ha añadido la opción "auth" directamente en el código. Una vez el cliente me proporcione su servidor SMTP, el *login* se realizará de una forma diferente (con OAuth2).

```
import Mail from 'nodemailer/lib/mailer';

export class MailSender {
    static async sendMail(mailOptions: Mail.Options) {
        const nodemailer = require('nodemailer');
        const transporter: Mail = nodemailer.createTransport({
            host: 'smtp.gmail.com',
            port: 465,
            secure: true, // true for 465, false for other ports
            auth: {
                user: 'gedisprod@gmail.com',
                pass: 'XXXXXX'
            }
        });
        // Lo usaremos para guardar todos los mails que enviemos
        mailOptions.from = mailOptions.from ? mailOptions.from : '"GeDisPro 👨‍⚕️" <gedisprod@gmail.com>';
        return await transporter.sendMail(mailOptions);
    }
}
```

Figura. 24 Mail Sender

6.1.3 *Pdf creator*

Para la creación de los PDF se ha usado la librería *pdfmake* la cual permite crear el documento a partir de un modelo creado por partes.

Por tanto, el script constará de varias partes, en primera instancia se crearán “Tables” con márgenes y estilos.

```
const dateInfo: Table = {
    headerRows: 1,
    widths: ['*', 80, 100, 100],
    heights: [20, 20],
    body: [
        [
            { border: [false, false, false, false], text: '' },
            { text: 'Ref. de pedido', fillColor: '#cccccc' },
            { text: 'Fecha de pedido', fillColor: '#cccccc' },
            { text: 'Fecha de entrega', fillColor: '#cccccc' }
        ],
        [
            { border: [false, false, false, false], text: '' },
            { border: [true, true, true, false], text: deliveryNote.ref },
            { border: [true, true, true, false], text: deliveryNote.dateCreation },
            { border: [true, true, true, false], text: deliveryNote.dateCloture }
        ]
    ]
};
```

Figura. 25 Tabla dateInfo

Todas las tablas creadas, serán añadidas al modelo dentro del cual se cargará un *background* predefinido dentro de la aplicación, dando a lugar al PDF final.

```
let docDefinition: TDocumentDefinitions = {
    background: [{ image: './resources/pdf/img/pdfbg.png', fit: [845, 845] }],
    pageMargins: [40, 120, 40, 120],

    content: [
        header,
        {
            table: deliveryNumber,
            margin: [0, 17, 0, 20]
        },
        {
            table: dateInfo
        },
        {
            table: clientInfo
        },
        {
            table: table,
            margin: [0, 0, 0, 50]
        },
        signContent
    ],

    styles
}
let pdf = printer.createPdfKitDocument(docDefinition);
pdf.pipe(fs.createWriteStream(`./resources/pdf/gen/${deliveryNote.file}.pdf`));
pdf.pipe(fs.createWriteStream(`./../web/src/assets/pdf/${deliveryNote.file}.pdf`));
pdf.pipe(fs.createWriteStream(`./../web/www/assets/pdf/${deliveryNote.file}.pdf`));
pdf.pipe(fs.createWriteStream(`./../web/platforms/browser/www/assets/pdf/${deliveryNote.file}.pdf`));
pdf.pipe(fs.createWriteStream(`./../web/platforms/ios/www/assets/pdf/${deliveryNote.file}.pdf`));
pdf.pipe(fs.createWriteStream(`./../web/platforms/android/app/src/main/assets/www/assets/pdf/${deliveryNote.file}.pdf`));
pdf.end();
}
```

Figura. 26 Creación PDF

6.2 Web

La infraestructura de la parte web difiere un poco de la anterior mencionada.

Dentro de la parte Web se encontrarán las carpetas con los módulos *node* utilizados. Por otro lado, existirá una carpeta con las plataformas habilitadas para crear la aplicación y todas sus SDKs. También se encontrará la carpeta con todos los *plugins* que se han utilizado de Cordova Apache.

Por último y más importante se encuentran los códigos y *assets* (iconos, imágenes…) que componen el *front*. Dentro de los códigos de la aplicación se podrán observar varias carpetas:

- Carpeta *core*: donde se almacenan todos los objetos, componentes e interfaces creados para ser utilizados globalmente en cualquier página de la aplicación.
- Carpeta *pages*: donde se encuentran cada uno de los códigos que conforman y crean las páginas de la aplicación. Cada página estará formada por 5 documentos diferentes:
    - Html: se escriben los códigos para dar formato a las páginas web.
    - SCSS: este documento sirve para dar estilos a los componentes creados en la página HTML.
    - Module: en el "module" se declararán las rutas, los *imports* y *exports* los cuales afectarán o se encontrarán en esta página.
    - TS: se declararán todas las funciones y acciones que será capaz de hacer la página.
    - Spec.ts: se declaran los *tests*.
- Carpeta *services*: los servicios, son los encargados de hacer la conexión con el servidor y hacer peticiones a los controladores vistos en el apartado superior.

Como explicar todos los códigos del Front puede ser muy tedioso, se explicará la implementación de las cosas más utilizadas y útiles dentro del *front* de la aplicación.

6.2.1 Login

Conforme avanza la informática, la delincuencia informática crece también, esto significa que cuando se va a desarrollar una aplicación se deben tener en cuenta qué brechas de seguridad se pueden encontrar. Una de las brechas principales de seguridad es la pantalla de *login*, donde pueden usar inyección de SQL para acceder a nuestra base de datos y así extraer nombres de usuarios y contraseñas de nuestros clientes.

Por eso a la hora de desarrollar el *login* se han adjuntado validadores pertenecientes a *angular/forms*, que no permiten que el texto introducido en la casilla del *email*, no sea un *email*.

Y por otro lado todas las contraseñas almacenadas en la base de datos están encriptadas con SHA256, por tanto, cuando se escriba una contraseña se comprobará la concordancia de los textos encriptados. Para la encriptación se ha usado la librería *crypto-js* perteneciente a *npm*, la cual contiene muchos estándares de encriptado.

```
// Encriptacion del password
const valueToSend = {
  email: this.form.value.email,
  password: CryptoJS.SHA256(this.form.value.password).toString(CryptoJS.enc.Hex).toUpperCase()
};
```

Figura. 27 Función de encriptado

6.2.2 Pantallas de creación de objetos

La reutilización de código es una práctica fundamental cuando se quiere hacer desarrollos de aplicaciones, esto nos lleva a la creación de interfaces y componentes, los cuales serán reutilizados en varias de las pantallas, una por comodidad ya que la creación de un componente nos permite no volver a escribir las mismas 20 líneas de código en cada pantalla.

Por este motivo para la creación de las pantallas que se utilizarán para insertar datos en la base de datos se han creado varias clases de *inputs*; *checkbox*, *date*, *dropdown*, *dropdown-multiple*… como también se ha creado una clase la cual contiene todos los tipos de mensajes tratados por estas páginas.

Por ejemplo, en la página de creación de usuarios las entradas de texto se crearán desde la página TypeScript de la manera siguiente:

```
this.users = [
  new InputText({ key: 'name', label: 'Nombre', order: 1, validators: [Validators.required] }),
  new InputText({ key: 'surname', label: 'Apellido', order: 2, validators: [Validators.required] }),
  new InputText({ key: 'email', label: 'Email', order: 3, validators: [Validators.required] }),
  new InputText({ key: 'password', label: 'Contraseña', order: 4, validators: [Validators.required] }),
  new InputText({ key: 'address_client', label: 'Dirección', order: 5, validators: [Validators.required] }),
  new InputDropdown({ key: 'fk_role', label: 'Rol', options: lstRoles, order: 6, validators: [Validators.pattern('[0-9]')] }),
];
```

Figura. 28 Utilización clases Input

Y para los *inputs* Dropdown nos hacemos servir de los servicios que se explicarán más abajo, para obtener la lista de todos los registros.

Para la creación de estas clases, nos haremos servir de interfaces donde se definirán las características que tendrán nuestras clases.

```
import { ValidatorFn } from '@angular/forms';
import { Option } from './option';

export interface IInputDropdown<T> {
    value?: T;
    key: string;
    label: string;
    order?: number;
    validators?: ValidatorFn[];
    options: Option[];
}
```

Figura. 29 Características de una interfaz

```
import { InputBase } from './input-base';
import { IInputBase } from '../interfaces/input-base';
import { IInputDropdown } from '../interfaces/input-dropdown';

export class InputDropdown extends InputBase<any> {
    constructor(options: IInputDropdown<any>) {
        const realOptions: IInputBase<any> = options;
        realOptions.controlType = 'dropdown';
        super(realOptions);
    }
}
```

Figura. 30 Input Dropdown

6.2.3 Servicios

La parte de los servicios es una de las más importantes de la aplicación ya que es la parte que nos proporciona la conexión con el servidor, proporcionando un servicio REST de respuestas JSON, las cuales se tratarán en el *front* de la aplicación para poder trabajar con los datos.

Un ejemplo de uno de los servicios de conexión sería el siguiente:

```
export class SubfamiliesService extends SuperService {
  constructor(private httpService: HttpClient) {
    super('http://192.168.18.2:3000/subfamily', httpService);
  }

  listByFamily(json: string): Promise<Message> {
    return this.http.post<Message>(`${this.url}/listByFamily`, json, this.httpOptions).toPromise();
  }
}
```

Figura. 31 Service family

Como se puede observar en la imagen anterior, todos los servicios extienden de un servicio llamado SuperService, el cual contiene funciones globales, las cuales se podrán usar con todos los servicios.

```
find(id: number): Promise<Message> {
  return this.http.get<Message>(`${this.url}/${id}`).toPromise();
}

delete(id: number): Promise<Message> {
  return this.http.delete<Message>(`${this.url}/${id}`, this.httpOptions).toPromise();
}

list(): Promise<any> {
  return this.http.get<Message>(`${this.url}/list`).toPromise();
}

create(json: string): Promise<Message> {
  return this.http.post<Message>(this.url, json, this.httpOptions).toPromise();
}

modify(id: number, json: string): Promise<Message> {
  return this.http.put<Message>(`${this.url}/${id}`, json, this.httpOptions).toPromise();
}
```

Figura. 32 Funciones del Superservice

La declaración de todas estas funciones dentro del Superservice, permite el ahorro de tiempo ya que no deberán ser añadidas a mano dentro de cada servicio, pudiendo declarar dentro de cada servicio las funciones específicas, si es necesario, de cada tabla.

7. Conclusiones y propuesta de trabajos futuros

Tras lo expuesto en la memoria, se a creado una aplicación capaz de tramitar pedidos, siendo accesible por los cinco actores diferentes y con funciones y acciones especificas para cada actor.

Uno de los puntos a destacar es la dificultad de crear una infraestructura aplicativa con componentes y interfaces, capaces de, en una fase avanzada de desarrollo, facilitar la escritura de código. Esto ha supuesto el tener que indagar en estas nuevas tecnologías y basarse en tutoriales de tecnologías parecidas para poder crear estas infraestructuras aplicativas capaces de proporcionar funcionalidades similares.

El hecho de haber trabajado con nuevas tecnologías, y con tecnologías que desde mi conocimiento profesional no había usado nunca, ha sido un reto y una satisfacción para mi persona, ya que me van ayudar en mi carrera profesional de forma muy positiva, pudiendo así en un futuro llevar a cabo proyectos más ambiciosos y de un mayor alcance.

Finalmente, quisiera hablar de mejoras futuras. Este proyecto ha sido desarrollado para una empresa conocida por el alumno, la cual ha proporcionado los datos de sus productos para poder crear una base de datos lo más real posible. Claro está que se han eliminado los precios de los productos ya que la empresa prefiere que sus precios solo sean conocidos por sus trabajadores y clientes. En cuanto al trabajo futuro, y como se podrá observar en algunas de las capturas de imágenes, se han empezado a desarrollar unas pantallas las cuales servirán para la otra empresa que han tenido que montar, ya que muchos clientes a los que proporcionaban productos de limpieza, les han demandado también si les pueden proporcionar máquinas y servicio técnico.

Por tanto, en los trabajos futuros y ya empezados a desarrollar se ha contemplado:

- Creación de interfaces, se creará una interfaz por cada tipo de máquina que proporcionen, ya que un horno no tiene las mismas características que un lavavajillas.
- Creación de tablas y páginas, para la creación de las máquinas proporcionadas por la empresa.
- Creación de una tabla y página donde se crearán y guardarán las incidencias que tengan en las máquinas y sean resueltas por los mecánicos.
- Creación de una página capaz de generar y leer códigos QR, que serán los identificadores de las máquinas.
- Adhesión de nuevas funcionalidades que demande la empresa para la parte de químicos.
- Creación de nuevas notificaciones para los mecánicos.

8. Bibliografía

[1] https://ionicframework.com/docs/

[2] http://ionicons.com/cheatsheet.html

[3] https://cordova.apache.org/docs/en/latest/

[4] https://angular.io/docs

[5] https://sqlite.org/docs.html

[6] https://blog.maestriajs.com/blog/ionic2/google-maps-native/

[7] https://pdfmake.github.io/docs/0.1/

[8] https://nodemailer.com/about/

[9] https://edupala.com/ionic-qr-code-scanner-and-generator/